**Московский государственный технический**

**Университет им. Н.Э. Баумана**

**Факультет «Радиотехничский»**

**Кафедра «Системы обработки информации и управления»**

Курс «Парадигмы и конструкции языков программирования»

Отчет по рубежному контролю №2

Выполнил:

студент группы РТ5-31Б

Фешин Д.А.

Проверил:

Гапанюк Е.Ю.

2024 г.

**Условия рубежного контроля №2 по курсу ПиК ЯП**

Рубежный контроль представляет собой разработку тестов на языке Python.

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

rk2.py

import unittest

from rk1 import Department, Employee, DepartmentEmployee, task1\_modified, task2\_modified, \

task3\_modified # Замените 'your\_module' на имя вашего файла

class TestTasks(unittest.TestCase):

def test\_task1\_filter\_departments\_with\_employees(self):

# Arrange (подготовка данных)

departments = [

Department(1, "Отдел разработки"),

Department(2, "Отдел маркетинга"),

Department(3, "Другой отдел")

]

employees = [

Employee(1, "Иванов Иван", 50000),

Employee(2, "Петрова Мария", 60000),

Employee(3, "Сидоров Антон", 70000)

]

department\_employees = [

DepartmentEmployee(1, 1),

DepartmentEmployee(1, 3),

DepartmentEmployee(2, 2),

]

# Act (вызов функции)

result = task1\_modified(departments, employees, department\_employees)

# Assert (проверки)

self.assertIn("Отдел разработки", result)

self.assertIn("Иванов Иван", result)

self.assertIn("Сидоров Антон", result)

self.assertNotIn("Отдел маркетинга", result)

def test\_task2\_calculate\_average\_salaries(self):

# Arrange

departments = [

Department(1, "Отдел разработки"),

Department(2, "Отдел маркетинга")

]

employees = [

Employee(1, "Иванов Иван", 50000),

Employee(2, "Петрова Мария", 60000),

Employee(3, "Сидоров Антон", 70000)

]

department\_employees = [

DepartmentEmployee(1, 1),

DepartmentEmployee(1, 3),

DepartmentEmployee(2, 2),

]

# Act

result = task2\_modified(departments, employees, department\_employees)

# Assert

self.assertIn("Отдел разработки: 60000.0", result)

self.assertIn("Отдел маркетинга: 60000.0", result)

def test\_task3\_find\_departments\_by\_employee\_name(self):

# Arrange

departments = [

Department(1, "Отдел разработки"),

Department(2, "Отдел маркетинга"),

Department(3, "Другой отдел")

]

employees = [

Employee(1, "Анна Иванова", 50000),

Employee(2, "Петрова Мария", 60000),

Employee(3, "Александр Сидоров", 70000)

]

department\_employees = [

DepartmentEmployee(1, 1),

DepartmentEmployee(2, 1),

DepartmentEmployee(3, 3),

]

# Act

result = task3\_modified(departments, employees, department\_employees)

# Assert

self.assertIn("Анна Иванова:", result)

self.assertIn("Отдел разработки", result)

self.assertIn("Отдел маркетинга", result)

self.assertIn("Александр Сидоров:", result)

self.assertIn("Другой отдел", result)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

rk1.py

class Department:

def \_\_init\_\_(self, id: int, name: str):

self.\_id = id

self.\_name = name

@property

def id(self):

return self.\_id

@property

def name(self):

return self.\_name

class Employee:

def \_\_init\_\_(self, id: int, name: str, salary: float):

self.\_id = id

self.\_name = name

self.\_salary = salary

@property

def id(self):

return self.\_id

@property

def name(self):

return self.\_name

@property

def salary(self):

return self.\_salary

class DepartmentEmployee:

def \_\_init\_\_(self, department\_id: int, employee\_id: int):

self.\_department\_id = department\_id

self.\_employee\_id = employee\_id

@property

def department\_id(self):

return self.\_department\_id

@property

def employee\_id(self):

return self.\_employee\_id

def task1\_modified(Departments, Employees, DepartmentEmployees):

print("Запрос 1 ")

data = [(d, [e for de in DepartmentEmployees for e in Employees if de.department\_id == d.id and de.employee\_id == e.id])

for d in Departments if "отдел" in d.name.lower()]

for department, employees in data:

print(department.name)

for employee in employees:

print(f" {employee.name}")

def task2\_modified(Departments, Employees, DepartmentEmployees):

print("Запрос 2 ")

data = {}

for department in Departments:

employees = [e for de in DepartmentEmployees for e in Employees if de.department\_id == department.id and de.employee\_id == e.id]

if employees:

avg\_salary = round(sum(e.salary for e in employees) / len(employees), 2)

data[department.name] = avg\_salary

for department, avg\_salary in sorted(data.items(), key=lambda x: x[1]):

print(f"{department}: {avg\_salary}")

def task3\_modified(Departments, Employees, DepartmentEmployees):

print("Запрос 3 ")

data = [(e, [d for de in DepartmentEmployees for d in Departments if de.department\_id == d.id and de.employee\_id == e.id])

for e in Employees if e.name.startswith("А")]

for employee, departments in data:

print(f"{employee.name}:")

for department in departments:

print(f" {department.name}")

def main():

# Создаем экземпляры классов для заполнения данными

departments = [

Department(1, "Отдел разработки"),

Department(2, "Отдел маркетинга"),

]

employees = [

Employee(1, "Иванов Иван Иванович", 50000),

Employee(2, "Петрова Мария Петровна", 60000),

]

department\_employees = [

DepartmentEmployee(1, 1), # Иванов в отделе разработки

DepartmentEmployee(2, 2), # Петрова в отделе маркетинга

]

# Вызываем функции для выполнения задач

task1\_modified(departments, employees, department\_employees)

task2\_modified(departments, employees, department\_employees)

task3\_modified(departments, employees, department\_employees)

if \_\_name\_\_ == "\_\_main\_\_":

main()